PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

Optical simulation for illumination using GPGPU ray tracing

Ryota Kimura, Masafumi Seigo, Russell A. Chipman, Seiichiro Kitagawa

Ryota Kimura, Masafumi Seigo, Russell A. Chipman, Seiichiro Kitagawa, "Optical simulation for illumination using GPGPU ray tracing," Proc. SPIE 10912, Physics and Simulation of Optoelectronic Devices XXVII, 109121A (26 February 2019); doi: 10.1117/12.2506129

SPIE.

Event: SPIE OPTO, 2019, San Francisco, California, United States

Optical simulation for illumination using GPGPU ray tracing

Ryota Kimura*[a], Masafumi Seigo*[a], Russell A. Chipman[b], Seiichiro Kitagawa[a]
[a]Nalux Co., Ltd., Yamazaki 2-1-7, Shimamoto-cho, Mishima-gun, Osaka 618-0001 Japan; [b] College of Optical Sciences, The University of Arizona, 1630 E. University Blvd., Tucson, AZ 85721-0094 U.S.A.

ABSTRACT

High-speed ray tracing for illumination optics using GPGPU was investigated. Optical simulation for illumination optics requires many rays tracing for precise simulation. Especially, optics for automotive LED lighting have small textures on the exit surface of the lens to diverge part of the light for satisfying specific illumination pattern which is required in the regulation. Many ray tracing requires much simulation times and it increases development cost. Recently, parallel computing using CPU and GPU has been used for accelerating computing speed and reported its merit in computer sciences. In this research, the ray tracing consists of two parts which are intersection searching and refraction calculation was done in parallel using CUDA, GPGPU API provided by NVIDIA. Interpolation calculations such as linear interpolation, Nagata triangular patch interpolation, and Nagata quadrilateral patch interpolation were used in intersection searching calculation. The results indicate that there is a possibility to accelerate ray tracing speed by using GPU. As a representative example, GPU ray tracing was about twice faster than the commercial software. In addition, error differences depend on the interpolation types for intersection calculation were observed. Moreover, the results indicate calculation error differences between single precision float calculation and double precision float calculation. In conclusion, even there are several issues such as errors from interpolation and calculation precision, accelerated ray tracing using GPU was achieved.

Keywords: ray tracing, CUDA, GPGPU, Nagata patch

1. INTRODUCTION

1.1 Ray Tracing

Ray tracing is basic calculation of geometrical optical simulation and important part of it. In general, rays propagate straight through homogeneous media and are bent by optical elements, lenses and mirrors and such large macroscopic structure.

1.2 Illumination optics for automotive

From about 10 years ago, advanced LED technology has been used to automotive headlamp. Since illumination distribution of the headlamp must satisfy regulations for traffic safety. Figure 1 shows an example of the light distribution of the headlamp.

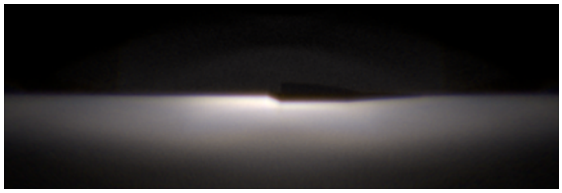


Figure 1. Light distribution of the headlamp

The regulations require not only light intensity at each points but also light gradation around cut line around horizontal line. It requires complex optics design and complex lenses. Figure 2 shows an actual LED projector type headlamp lens.

Physics and Simulation of Optoelectronic Devices XXVII, edited by Bernd Witzigmann, Marek Osiński, Yasuhiko Arakawa, Proc. of SPIE Vol. 10912, 109121A · © 2019 SPIE
CCC code: 0277-786X/19/$18 · doi: 10.1117/12.2506129

Figure 2. Headlamp lens with tiny texture

It has tiny textures on the exiting surface of the lens to diffuse a part of light for reducing condense of the light. Since the textures are tiny, enough number of incident rays are required for precise simulation. Therefore, optics for LED headlamp is complex design and its simulation requires massive calculation power.

2. RAY TRACING

2.1 Intersection Point Searching

Intersection searching is first part of ray tracing and can be solved mathematically. In this paper, intersection between a ray and a plane, a ray and a sphere, and a ray and point cloud are introduced.

2.1.1 Intersection Point with Plane

The simplest object for intersection with a ray is a plane surface. Figure 3 shows calculation model for intersection searching between a ray and a plane. The ray in three-dimensional space can be described as,

$$x = x_0 + tk_x, \tag{1}$$

$$y = y_0 + tk_y, \tag{2}$$

$$z = z_0 + tk_z, \tag{3}$$

where $\{x_0, y_0, z_0\}$ is the initial position of the ray. t is distance from $\{x_0, y_0, z_0\}$. $\{k_x, k_y, k_z\}$ is a unit directional vector. The plane surface that contains a point {a, b, c} can be described as,

$$n_x(x-a) + n_y(y-b) + n_z(z-c) = 0, \tag{4}$$

where $\{n_x, n_y, n_z\}$ is a normal vector of the plane surface. Equation (1), (2), and (3) can be substituted into equation (4) and solved for t,

$$t = \frac{-n_x x_0 + n_x a - n_y y_0 + n_y b - n_z z_0 + n_z c}{n_x k_x + n_y k_y + n_z k_z}. \tag{5}$$

From t, the intersection point {x, y, z} can be derived by substitution of t into equation (1), (2), and (3).

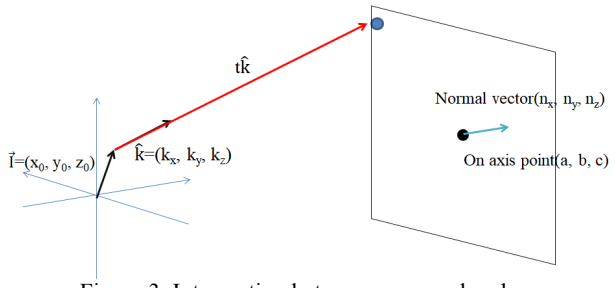


Figure 3. Intersection between a ray and a plane

2.1.2 Intersection Point with Sphere

The intersection between a ray and a spherical surface can be determined as follows. Figure 4 shows calculation model for intersection searching between a ray and a sphere. **p** is vector from the center of the sphere to the surface of the

sphere,

$$|\vec{p}| = r, \tag{6}$$

where r is the radius of the spherical surface. Furthermore,

$$|\vec{p}| = -\vec{s} + \vec{I} + t\vec{k}, \tag{7}$$

where **s** is a vector from the origin of the space to the origin of the sphere and **I** is an initial position of the ray, t is a constant, and **k** is a directional vector of the ray. Substitute equation (6) into equation (7) and take square of both sides,

$$t^2\left(\left|\vec{k}\right|^2\right) + t\left(-2\vec{s}\cdot\vec{k} + 2\vec{k}\cdot\vec{I}\right) + \left(|\vec{s}|^2 - 2\vec{s}\cdot\vec{I} + \left|\vec{I}\right|^2 - r^2\right) = 0. \tag{8}$$

Since equation (8) is a quadratic function of t,

$$\mathrm{t} = \frac{-\left(-2\vec{s}\cdot\vec{k} + 2\vec{k}\cdot\vec{I}\right) \pm \sqrt{\left(-2\vec{s}\cdot\vec{k} + 2\vec{k}\cdot\vec{I}\right)^2 - 4\left(\left|\vec{k}\right|^2\right)\left(|\vec{s}|^2 - 2\vec{s}\cdot\vec{I} + \left|\vec{I}\right|^2 - r^2\right)}}{2\left(\left|\vec{k}\right|^2\right)}. \tag{9}$$

The equation indicates that t has two answers. The smaller t is for the closer intersection points which are on the front surface of the sphere, and the larger is for farther intersection on the back surface of the sphere. If t is an imaginary number, it indicates that the ray does not intersect with the spherical surface but misses.

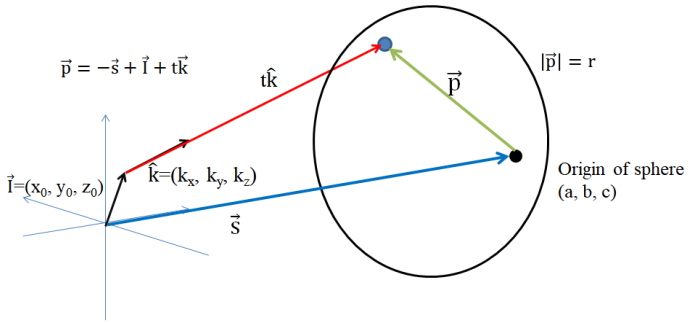


Figure 4. Intersection between a ray and a sphere

2.1.3 Intersection Point with Complex Surface

Actual surface models in ray tracing are often more complex surfaces than plane surfaces and spherical surfaces. In this study, point clouds are used to express complex shaped surfaces because these are often used to communicate complex shapes from CAD programs. Each point has a three-dimensional coordinates and an associated normal vector. The points are addressed as an array and express the complex shaped surfaces. To find the Intersection point between a ray and the surface, **p** which is a vector from the initial point to a point in the cloud is compared with a propagation vector **k**. If the **p** points the same direction with **k,** the ray hit the point. Mathematically, evaluation value f can be calculated by cross product as,

$$\mathrm{f} = \frac{\overrightarrow{\mathrm{p_{mn}}} \times \hat{k}}{\left|\overrightarrow{\mathrm{p_{mn}}}\right|} = \sin\theta_{\mathrm{p_{mn}k}}, \tag{10}$$

where $\mathbf{p}_{mn}$ is the vector from the initial point to the point located at m^{th} in horizontal and n^{th} in vertical in the point cloud. θ_{pmnk} is the angle between $\mathbf{p}_{mn}$ and **k**. In the simplest algorithm, the evaluation values for all points in the point cloud are evaluated with a brute force attack. The point which has the smallest evaluation value is the nearest point from the intersection point between the ray and the surface. When the evaluation value is not zero, interpolation using points in the nearby points are required. A variety of interpolation methods are described in the following. Since this evaluation method requires a calculation for all points, much unnecessary calculation is performed. If a point cloud consists of 100 by 100 points, the calculation of the nearest point requires 10,000 loops. To reduce calculation loops, several method is used. In this project, k-d tree is used. Figure 6 shows calculation model of k-d tree. Number of candidate points are reduced by half in each step. The required loop number can be estimated as,

$$2^{\mathrm{Q}} = \mathrm{M}, \tag{11}$$

where, Q is required loop number and M is the number of the points in a cloud. The equation solves,

$$\mathrm{Q} = \frac{\log_{10}\mathrm{M}}{\log_{10}2}, \tag{12}$$

Therefore, calculation time is reduced dramatically over the brute force.

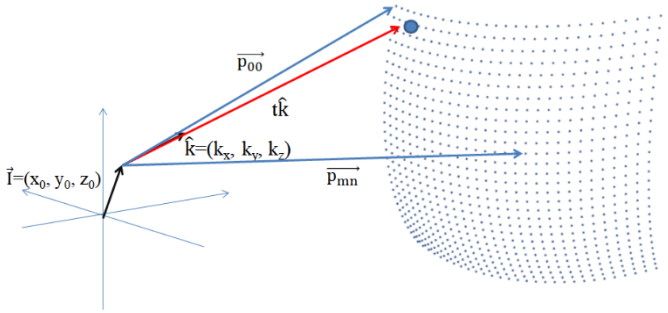


Figure 5. Intersection between a ray and a complex surface

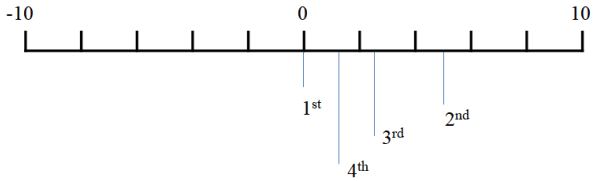


Figure 6. K-d tree example

After finding nearest intersection points, interpolation is required for acquiring intersection point and surface normal vector at the intersection point. Simplest interpolation is linear interpolation from vicinity points. Intersection is calculated from vicinity points and a plane surface consists of the vicinity points. Nagata et al. introduced a simple quadratic interpolation algorithm from vicinity points and normal vectors which is known as the Nagata patch. [1, 2] Following is only simple results of Nagata patch. Figure 7 shows Nagata triangular patch. **X** is vicinity point location. **n** is normal vector at each vicinity points. η and ζ are localized unit vector.
The quadratic interpolated point is described as,

$$\vec{X}(\eta,\xi) = \vec{X}_{00}(1-\eta) + \vec{X}_{10}(\eta-\xi) + \vec{X}_{11}\xi - \vec{c}_1(1-\eta)(\eta-\xi) - \vec{c}_2(\eta-\xi)\xi - \vec{c}_3(1-\eta)\xi \ . \tag{13}$$

$\mathbf{c}_1$, $\mathbf{c}_2$ and $\mathbf{c}_3$ are curvature parameters and defined as,

$$\vec{c}_1 \equiv \vec{c}(\vec{d}_1, \vec{n}_{00}, \vec{n}_{10}) \ , \tag{14}$$

$$\vec{c}_2 \equiv \vec{c}(\vec{d}_2, \vec{n}_{10}, \vec{n}_{11}) \ , \tag{15}$$

$$\vec{c}_3 \equiv \vec{c}(\vec{d}_3, \vec{n}_{00}, \vec{n}_{11}) \ . \tag{16}$$

where, **c** is function and defined as,

$$\vec{c}_m(\vec{d}_m, \vec{n}_a, \vec{n}_b) = \begin{cases} \dfrac{\Delta d_m}{1-\Delta c}\vec{\nu} + \dfrac{d}{\Delta c}\Delta\vec{\nu} & (c \neq \pm 1) \\ \{0,0,0\} & (c = \pm 1). \end{cases} \tag{17}$$

where ν is an average of the normal vectors and Δν is deviation of the normal vectors.

$$\vec{\nu}_{ab} = (\vec{n}_a + \vec{n}_b)/2 \ , \tag{18}$$

$$\Delta\vec{\nu}_{ab} = (\vec{n}_a - \vec{n}_b)/2 \ . \tag{19}$$

d and Δ are defined as,

$$d_m = \vec{d}_m{}^T \vec{\nu}_{ab} \ , \tag{20}$$

$$\Delta d_m = \vec{d}_m{}^T \Delta\vec{\nu}_{ab} \ . \tag{21}$$

Each **d** is defined as,

$$\vec{d}_1 = \vec{X}_{10} - \vec{X}_{00} \ , \tag{22}$$

$$\vec{d}_2 = \vec{X}_{11} - \vec{X}_{10} \ , \tag{23}$$

$$\vec{d}_3 = \vec{X}_{11} - \vec{X}_{00} \ . \tag{24}$$

c and Δc are defined as

$$c_{ab} = 1 - 2\Delta c_{ab} \ , \tag{25}$$

$$\Delta c_{ab} = \vec{n}_a{}^T \Delta \vec{v}_{ab} \ . \tag{26}$$

In addition, the normal vectors at the interpolated point can be calculated by taking partial derivative of **X** respect to η and ξ as,

$$\vec{X}_\eta = \frac{\partial \vec{X}}{\partial \eta} = \vec{d}_1 + \vec{c}_1\{(\eta - \xi) - (1 - \eta)\} + (\vec{c}_3 - \vec{c}_2)\xi \ , \tag{27}$$

$$\vec{X}_\xi = \frac{\partial \vec{X}}{\partial \xi} = \vec{d}_2 + \vec{c}_2\{\xi - (\eta - \xi)\} + (\vec{c}_1 - \vec{c}_3)(1 - \eta) \ . \tag{28}$$

The normal vector at η and ξ are calculated as,

$$\vec{n}(\eta, \xi) = \frac{\partial \vec{X}}{\partial \eta} \times \frac{\partial \vec{X}}{\partial \xi} \ , \tag{29}$$

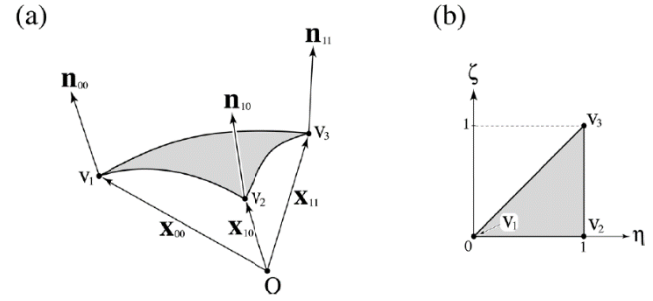


Figure 7. Nagata triangular patch [1]

Similarly, they introduced quadrilateral patch method as figure 8. Equation for quadrilateral patch is omitted here.

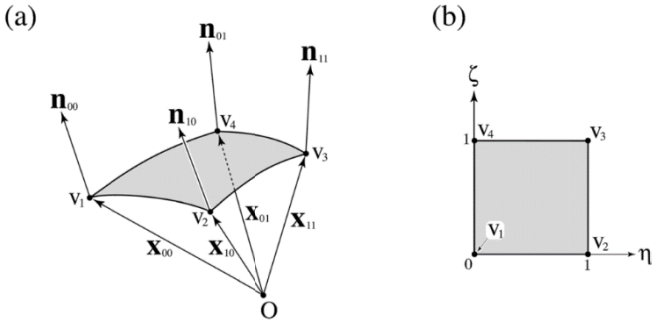


Figure 8. Nagata quadrilateral patch [1]

2.2 Refraction Calculation

Objects and rays are defined in three-dimensional space for ray tracing, so Snell's law is expanded into three-dimensional space. In figure 9, an incident ray unit vector is shown as **i**. A unit refracted exiting ray vector is shown as **r**. **n** shows the unit normal vector at the boundary. **a** and **b** are orthogonal unit vectors with **n** and orthogonal each other. From trigonometry,

$$\hat{r} = \hat{a} \sin\theta' - \hat{n} \cos\theta'. \tag{30}$$

Since a, b, and c are orthogonal,

$$\hat{a} = \hat{n} \times \hat{b} = \hat{n} \times \left(\frac{\vec{i} \times \hat{n}}{\sin\theta}\right) = \frac{1}{\sin\theta}\{(\hat{n} \cdot \hat{n})\hat{i} - (\hat{n} \cdot \hat{i})\hat{n}\} = \frac{1}{\sin\theta}\{\hat{i} - (\hat{n} \cdot \hat{i})\hat{n}\}. \tag{31}$$

Substitute Snell's law into (31),

$$\hat{a} = \frac{n}{n' \sin\theta'}\{\hat{i} - (\hat{n} \cdot \hat{i})\hat{n}\}. \tag{32}$$

From trigonometry,

$$\cos\theta = -\hat{i} \cdot \hat{n} \ . \tag{33}$$

From Pythagoras's identity,

$$\sin^2\theta = 1 - \cos^2\theta = 1 - (-\hat{i} \cdot \hat{n})^2 \ . \tag{34}$$

Substitute Snell's law into (34),

$$\left(\frac{n'}{n}\right)^2 \sin^2\theta' = 1 - (-\hat{i} \cdot \hat{n})^2 \ . \tag{35}$$

$$\sin^2\theta' = \left(\frac{n}{n'}\right)^2 \left\{1 - (-\hat{i} \cdot \hat{n})^2\right\} . \tag{36}$$

From Pythagoras's identity,

$$\cos^2\theta' = 1 - \left(\frac{n}{n'}\right)^2 \left\{1 - (-\hat{i} \cdot \hat{n})^2\right\}, \tag{37}$$

$$\cos\theta' = \sqrt{1 - \left(\frac{n}{n'}\right)^2 \left\{1 - (-\hat{i} \cdot \hat{n})^2\right\}}. \tag{38}$$

Substitute (2.50) and (2.56)into (2.48),

$$\begin{aligned}
\hat{t} &= \frac{n}{n' \sin\theta'}\{\hat{i} - (\hat{n} \cdot \hat{i})\hat{n}\} \sin\theta' - \hat{n}\sqrt{1 - \left(\frac{n}{n'}\right)^2 \left\{1 - (-\hat{i} \cdot \hat{n})^2\right\}} \\
&= \frac{n}{n'}\{\hat{i} - (\hat{n} \cdot \hat{i})\hat{n}\} - \hat{n}\sqrt{1 - \left(\frac{n}{n'}\right)^2 \left\{1 - (-\hat{i} \cdot \hat{n})^2\right\}} \\
&= \frac{n}{n'}\hat{i} - \frac{n}{n'}\left[(\hat{n} \cdot \hat{i}) + \sqrt{\left(\frac{n'}{n}\right)^2 - \left\{1 - (-\hat{i} \cdot \hat{n})^2\right\}}\ \right]\hat{n} \\
&= \frac{n}{n'}\hat{i} - \frac{n}{n'}\left[(\hat{n} \cdot \hat{i}) + \sqrt{\left(\frac{n'}{n}\right)^2 - 1 + (\hat{n} \cdot \hat{i})^2}\ \right]\hat{n}
\end{aligned} \tag{39}$$

Now, refracted vector **r** is described with refractive indices, the incident vector, and the surface normal vector. When the square root in equation (39) is negative, the ray cannot refract at the boundary, so the ray is reflected.

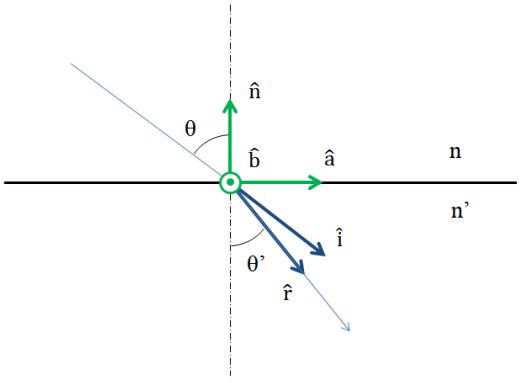


Figure 9. Snell's law in three dimension

3. GPGPU

3.1 CUDA

NVIDIA describes CUDA as “a general purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU.”[3] CUDA consists of a driver, a compiler, and APIs. The driver is interface software which connects a device to the system for use in the operating system. The compiler transforms program code into machine code. The CUDA compiler is compatible with the C and C++ and expanded for calculation in GPU. APIs provide program sets which makes it easy to programming effectively. In CUDA, code running on a single core is called a thread. CUDA manages several threads as a group called a block. Several blocks are managed as a grid. The block and the grid can be set in one dimension, two dimensions, or three dimensions when the CUDA program is launched. Its maximum size depends on the GPU specification. The configuration of block and grid should be considered for effective processing in CUDA. The purpose of CUDA is to provide functions that allow for parallel computing.

4. SIMULATION RESULT

4.1 Simulation Model

To validate the program, following plano-convex aspherical lens is simulated. For CUDA ray tracing, first aspherical lens is expressed as a point cloud which point pitch is 0.05mm and consists of 78,961 points. Second and detector surface is expressed as a plane surface. Diameter of the lens is 12mm. The focal plane position is optimized for minimization of spot radius located at 23.168mm from plane surface. Figure 10 shows simulation model and simulation result from OpticStudio which is commercial optical simulation software. Since the aspherical surface is optimized to minimal spot size, the spot size at the focal plane nearly zero.

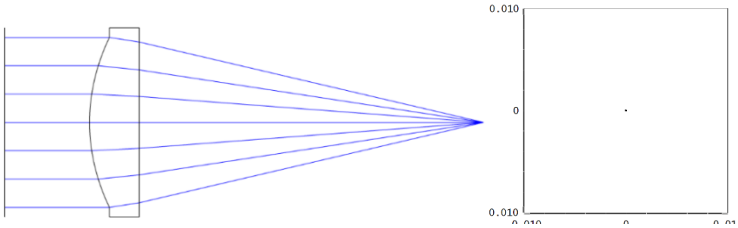


Figure 10. Simulation model for validation with OpticStudio simulation result

4.2 Simulation Time Comparison

As expectation, GPGPU accelerate ray tracing calculation speed. Figure 11 shows calculation time comparison of the ray tracing with Nagata triangular patch. The test was done with a PC which consists with Core i7-8700K(3.7GHz), 16GB RAM, GeForce GTX1080 (2560 CUDA Core, 8GB GDDR5X RAM), and windows 10. First result is from single thread CPU calculation with double float precision. Second result is from multi thread CPU calculation with double float precision using OpenMP. Third is result from double float precision calculation in GPU. Fourth is result from single float precision calculation in GPU. Since GPU cannot access main memory directly, GPGPU requires GPU driver initialization, data copying from main memory to GPU memory and result data copying from GPU memory to main memory. When the ray number for simulation is much, GPGPU take time advantage even the data copying takes few times. In this calculation condition, CUDA calculation is about twice faster than CPU calculation.

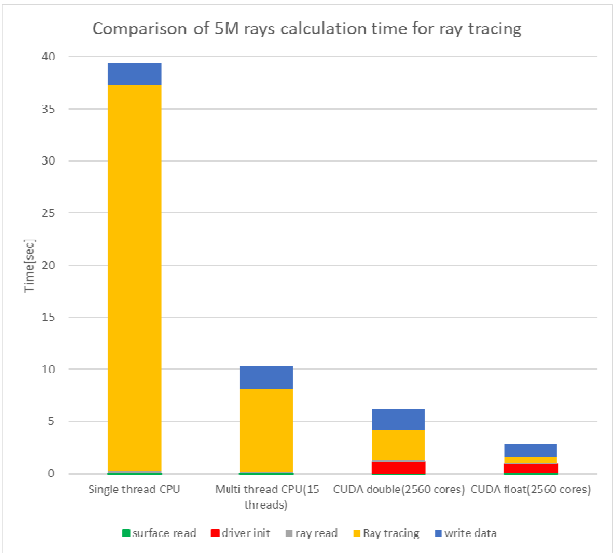
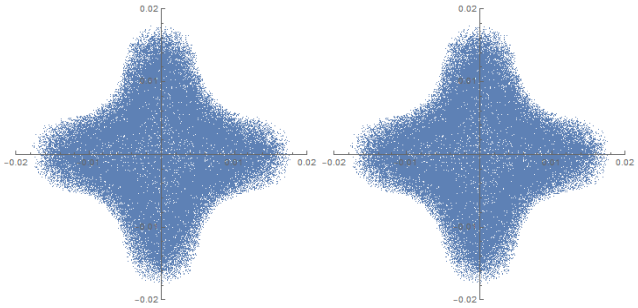


Figure 11. Time comparison of CPU calculation and GPU calculation

4.3 Simulation Accuracy

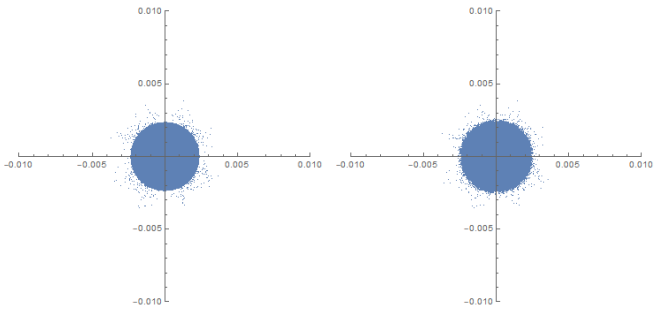
There are some errors in the results and the errors depends on the interpolation types and float precision. Figure 12 shows the spot diagrams at the detector surface of CUDA ray tracing with linear interpolation. The error is about ±17.5μm in both x and y direction.



Double precision float Single precision float

Figure 12. Spot diagram of an aspherical lens using CUDA ray tracing (Linear interpolation)

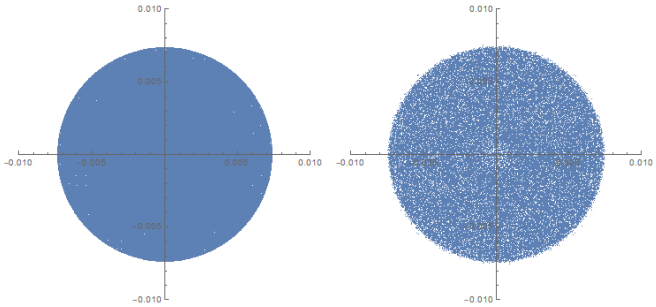
Figure 13 shows the spot diagrams of CUDA ray tracing with Nagata triangular patch interpolation. The error is about ±2.3μm in both x and y direction.



Double precision float Single precision float

Figure 13. Spot diagram of an aspherical lens using CUDA ray tracing (Nagata triangular patch)

Figure 14 shows the spot diagrams of CUDA ray tracing with Nagata quadrilateral patch interpolation. The error is about ±7.2μm in both x and y direction.



Double precision float Single precision float

Figure 14. Spot diagram of an aspherical lens using CUDA ray tracing (Nagata quadrilateral patch)

5. CONCLUSION

Several types of intersection searching methods and interpolation methods include Nagata patch were reviewed. Then, refraction calculations in 3-dimensional space were reviewed. In addition, CUDA, GPGPU was reviewed. Ray tracing results using each interpolation method were described, and their accuracies were discussed. Validation test was used a plano-convex aspherical lens. Results indicate that the double float precision Nagata triangular patch interpolation is the most accurate. In conclusion, an accelerated ray tracing method using CUDA for models with spherical surfaces, plane surfaces, and point clouds with linear and quadratic interpolation have been developed in this study. High-speed ray tracing was achieved. However, some errors were documented from interpolation.

REFERENCES

[1] Takashi Nagata, “Simple local interpolation of surfaces using normal vectors,” Computer Aided Geometric Design 22, 327-347 (2005)

[2] Shinya Morita, Yohei Nishidate, Takashi Nagata, Yutaka Yamagata, and Cristian Teodosiu. "Ray-tracing Simulation Method Using Piecewise Quadratic Interpolant for Aspheric Optical Systems." *Applied Optics* 49, no. 18 (2010).

[3] NVIDIA Corporation. ”CUDA C Programming Guide (v8.0),” http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[4] E. L. Dereniak, T. D. Dereniak, “Geometrical and trigonometric optics. Cambridge,” UK;New York;: Cambridge University Press (2008).